ORACLE NOSQL DATABASE HANDS-ON WORKSHOP – Cluster Deployment and Management

ORACLE

Lab Exercise 1 – Deploy 3x3 NoSQL Cluster into single Datacenters

Objective:

Learn from your experience how simple and intuitive it is to deploy a highly available (production ready) Oracle NoSQL Database cluster.

Summary:

We will use Big Data Lite VM for all the labs mentioned in this document. This VM besides many other Big Data software has Oracle NoSQL Database binaries and all the helper scripts that we are going to use in all the labs are preinstalled. If you just want to take a look at the scripts then you can download it from here.

After we are done with lab -1 here is how our topology is going to look like.

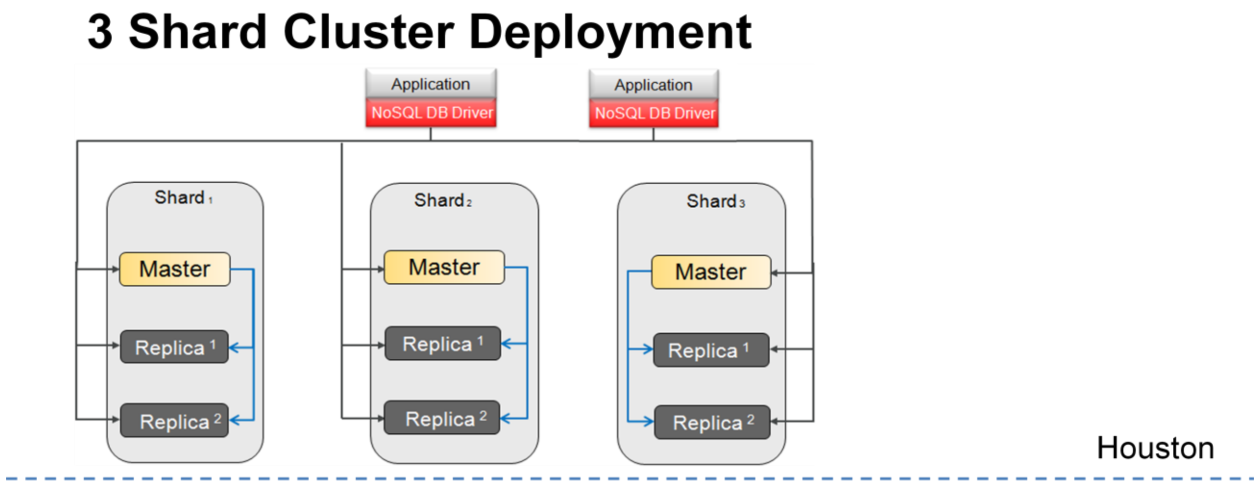


Figure 1: Three shard cluster topology deployed in primary datacenter (Houston) with replication factor (RF)=3.

The cluster topology is going to have three shards running in Houston datacenter, with each shard having three replication nodes (RF=3). So total of 9 nodes will be deployed on same virtual machine.

Instructions:

First make sure you are logged into VM as user 'oracle'.:

1. Open a terminal window and type 'echo $KVHOME'. This is where Oracle NoSQL Database binaries are staged.

2. Next change directory to */u01/nosql/scripts* and list files in that directory. We will be using these scripts for cluster deployment

```
[oracle@bigdatalite ~]$ cd /u01/nosq/scripts

[oracle@bigdatalite scripts]$ ls

3x3.kvs      deleteDir.sh          runScript.sh       tables
4x3.kvs      makebootconfig.sh     showUpgradeOrder.sh  upgradeSNA.sh
4x4.kvs      makebootconfig-sn4.sh  startSNA.sh
admin.sh     makebootconfig-sn5.sh  startSNA-sn4.sh
createDir.sh  resetEnv.sh          startSNA-sn5.sh
```

3. Before we use any physical machine to host NoSQL Database instances, we need to bootstrap the machines. Let's take a look at makebootconfig.sh and learn how we are bootstrapping three storage nodes on this single virtual machine. Type "vi makebootconfig.sh" from the terminal window and review the script.

```
[oracle@bigdatalite scripts]$ vi makebootconfig.sh


########### Bootstrap Storage Node 1 ##########
java -jar $KVHOME/lib/kvstore.jar makebootconfig \
        -root $SN1_DATA_HOME/kvroot \
        -store-security none \
        -capacity 3 \
        -harange 5010,5030 \
        -admin 5001 \
        -port 5000 \
        -memory_mb 200\
        -host kvhost01 \
        -storagedir $SN1_DATA_HOME/u01 \
        -storagedir $SN1_DATA_HOME/u02 \
        -storagedir $SN1_DATA_HOME/u03 \


…
```

Notice that we are defining three different port (-admin, -harangue, -port) ranges for various inter-intra node communication. Number of replication-nodes (RN) this storage node can host is defined by the capacity as '*-capacity*'. You can define storage directories where data is going to be written using -*storagedir* argument.

In our case we defined capacity=3, which means we would like three RNs to be hosted on each of the three storage node with each RN using its own disk. This is important aspect as independent spindles improves performance and the availability of the system.

4. Let's run this *makebootconfig.sh* script.

```
[oracle@bigdatalite scripts]$ ./makebootconfig.sh

Killing Java processes ...
kill 7524: No such process
Deleting old directories...
Creating new directories...
 Done bootstrapping storage-node 1
 Done bootstrapping storage-node 2
 Done bootstrapping storage-node 3
```

You would notice that the wrapper script will clean the environment for us, and also bootstrap three storage nodes by writing bunch of config files under '-root' directory. If you want you can list any of the KVROOT to find out what is in there.

5. Once all the three storage node (physical machine) are boot strapped we are going to start Storage Node Agents (SNA) on each of the SN, which will act as a listeners and a resource manager. Before we start let's take a look at the *startSNA.sh* script. Just do cat on the file.

```
[oracle@bigdatalite scripts]$ cat startSNA.sh

java -jar $KVHOME/lib/kvstore.jar start -root /tmp/data/sn1/kvroot &
java -jar $KVHOME/lib/kvstore.jar start -root /tmp/data/sn2/kvroot &
java -jar $KVHOME/lib/kvstore.jar start -root /tmp/data/sn3/kvroot &
```

Notice that we are passing KVROOT of the SN to the start command.

6. Now execute the *startSNA.sh* script to start the agent.

```
[oracle@bigdatalite scripts]$ ./startSNA.sh
```

7. Now if we run jps command then we can see three ‘kvstore.jar’ instances and three ‘ManagerService’ instances running:

```
[oracle@bigdatalite scripts]$ jps
8756 ManagedService
8940 Jps
8561 kvstore.jar
8560 kvstore.jar
8718 ManagedService
8559 kvstore.jar
8762 ManagedService
```

8. Next we are going to run admin shell script to deploy 3x3 topology on storage nodes that we just started in previous step. Let’s take a look at the 3x3.kvs script first:

```
[oracle@bigdatalite scripts]$ vi 3x3.kvs
### Begin Script ##########################################
configure -name kvstore
plan deploy-datacenter -name "Houston" -rf 3 -wait

####### Deploy First Storage Node  #########################
plan deploy-sn -dcname "Houston" -port 5000 -wait -host kvhost01
plan deploy-admin -sn sn1 -port 5001 -wait

###### Deploy Second Storage Node ##########################
plan deploy-sn -dcname "Houston" -port 6000 -wait -host kvhost02
plan deploy-admin -sn sn2 -port 6001 -wait

###### Deploy Third Storage Node ###########################
plan deploy-sn -dcname "Houston" -port 7000 -wait -host kvhost03
plan deploy-admin -sn sn3 -port 7001 -wait

####### Create topology, preview it and then deploy it #####
topology create -name 3x3 -pool AllStorageNodes -partitions 120
topology preview -name 3x3
plan deploy-topology -name 3x3 -wait
### End Script ############################################
```

Notice that we first create a store by name 'kvstore' and then we deploy a datacenter Houston. In this datacenter we then create three storage nodes and deploy admin servers on each of the storage node.

After that we create a topology 3x3 and assign all the existing resources to that topology. Next we preview and then deploy the cluster.

9. Let's take a look at runadmin.sh script that we will use to run the 3x3.kvs script. This script takes an input argument (using *load -file* option) which is the name of the admin shell script

```
[oracle@bigdatalite scripts]$ cat runScript.sh
java -jar $KVHOME/lib/kvstore.jar runadmin -port 5000 -host kvhost01 load -file $1
```

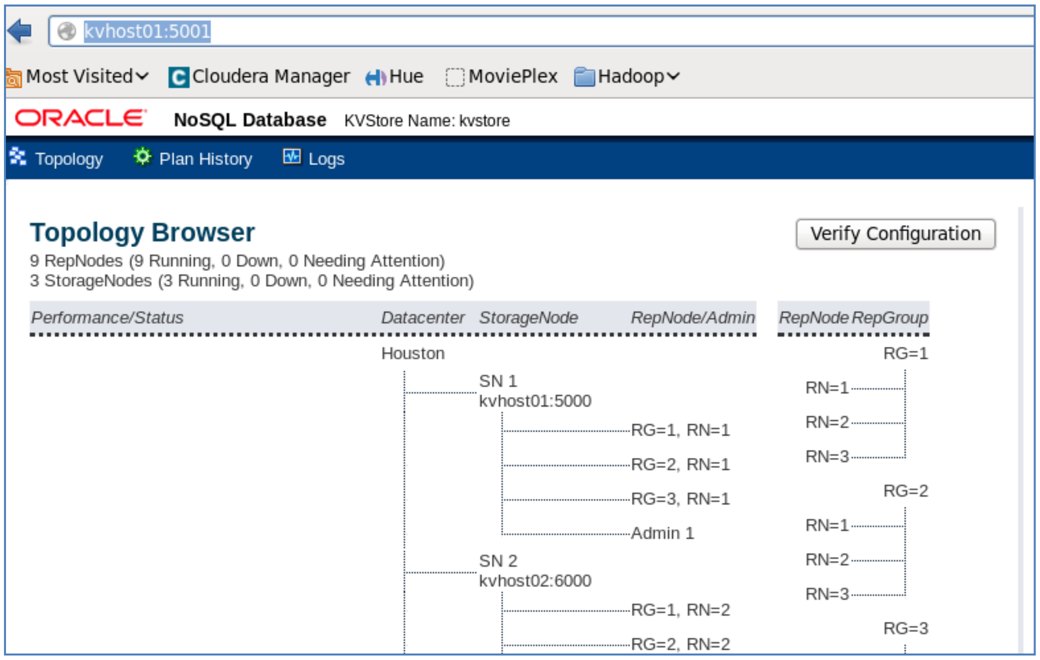
10. We are now ready to deploy the 3x3 topology using 'runadmin.sh 3x3.kvs' command:

```
[oracle@bigdatalite scripts]$ ./runScript.sh 3x3.kvs
Store configured: kvstore
Executed plan 1, waiting for completion...
Plan 1 ended successfully
....
Created: 3x3
Topology transformation from current deployed topology to 3x3:
Create 3 shards
Create 9 RNs
Create 120 partitions

shard rg1
  3 new RNs : rg1-rn1 rg1-rn2 rg1-rn3
  40 new partitions
shard rg2
  3 new RNs : rg2-rn1 rg2-rn2 rg2-rn3
  40 new partitions
shard rg3
  3 new RNs : rg3-rn1 rg3-rn2 rg3-rn3
  40 new partitions
Executed plan 8, waiting for completion...
Plan 8 ended successfully
```

Notice that the preview command is informing how many shards, replica-nodes (RNs) and partitions are going to be created.

You all can also open ***http://kvhost01:5001/*** to view the topology so far from the web console:



Topology Browser

9 RepNodes (9 Running, 0 Down, 0 Needing Attention)
3 StorageNodes (3 Running, 0 Down, 0 Needing Attention)

```
Performance/Status    Datacenter  StorageNode     RepNode/Admin     RepNode RepGroup
                      Houston                                                  RG=1
                                  SN 1                                RN=1
                                  kvhost01:5000                       RN=2
                                                  RG=1, RN=1          RN=3
                                                  RG=2, RN=1                   RG=2
                                                  RG=3, RN=1          RN=1
                                                  Admin 1             RN=2
                                  SN 2                                RN=3
                                  kvhost02:6000                                RG=3
                                                  RG=1, RN=2
                                                  RG=2, RN=2
```

Figure 2: Web based administration tool showing cluster topology and the state of replication nodes under each shard.

Lab Exercise 2 – Expand NoSQL Database cluster by adding a new Shard.

Objective:

Learn how simple it is to expand NoSQL Database cluster on demand without affecting the availability of any shard. Our goal is to expand 3x3 topology to 4x3 (i.e. 4 shards with each shard having three replicas).

Summary:

In previous labs we learned how to deploy fault tolerant, highly available Oracle NoSQL cluster in a distributed environment. In this lab we will spend some time learning how to scale the cluster by adding more storage nodes.

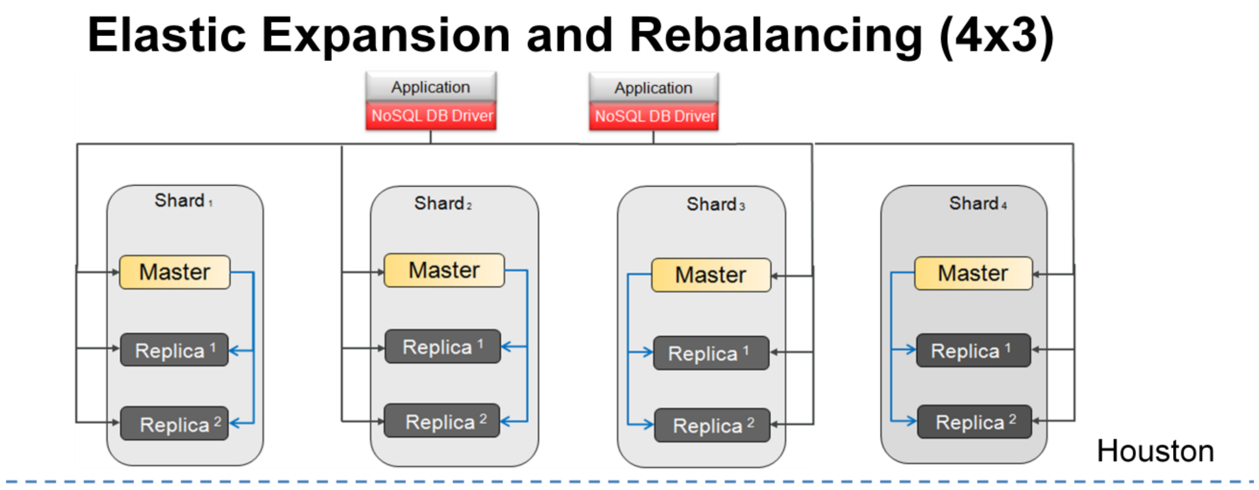


Figure 3: On demand cluster expansion from the 3x3 topology to 4x4 topology.

By the end of the second lab we are going to have four shards in Houston data center.

Instructions:

Like in previous lab, we will use helper scripts to drive this lab as well (just to prevent the typos).

1. In production environment you would be adding new physical machines to expand your NoSQL cluster but we have a single VM only to work with, so we are going to emulate similar condition by creating a new storage node from different KVROOT directory and using different ports. Let's view and run makebootconfig-sn4.sh script:

```
[oracle@bigdatalite scripts]$ cat ./makebootconfig-sn4.sh
#!/bin/bash

SN4_DATA_HOME=/tmp/data/sn4

########### Bootstrap Storage Node 4 ##########
java -jar $KVHOME/lib/kvstore.jar makebootconfig \
        -root $SN4_DATA_HOME/kvroot \
        -capacity 3 \
        -harange 8010,8030 \
        -admin 8001 \
        -port 8000 \
        -memory_mb 200\
        -host kvhost04 \
        -storagedir $SN4_DATA_HOME/u01 \
        -storagedir $SN4_DATA_HOME/u02 \
        -storagedir $SN4_DATA_HOME/u03 \

echo " Done bootstrapping storage-node 4"
#####################################################
```

   Note: We are using 8000 port range for this SN where previous three SN used 5000, 6000 & 7000. Also the storagedir option is used to store the data under SN4 storage directory. In real production environment you would be hosting single SN per physical node so you don't have to change the ports across multiple SNs.

2. Run the makebootconfig-sn4.sh script to bootstrap SN4

```
[oracle@bigdatalite scripts]$ ./makebootconfig-sn4.sh
 Done bootstrapping storage-node 4
```

3. Next start the storage agent on newly added storage node:

```
[oracle@bigdatalite scripts]$ cat startSNA-sn4.sh
java -jar $KVHOME/lib/kvstore.jar start -root /tmp/data/sn4/kvroot &
```

```
[oracle@bigdatalite scripts]$ ./startSNA-sn4.sh
```

4. Next we will review 4x3.kvs script. We just need to add new storage node to Houston datacenter and then based on the current state of the cluster we clone the topology and call redistribute command, which will create new shards, replication nodes and move partitions, RN to rebalance the cluster.

```
[oracle@bigdatalite scripts]$ cat 4x3.kvs
###### Deploy Fourth Storage Node  #########################
plan deploy-sn -dcname "Houston" -port 8000 -wait -host kvhost04

topology clone -current -name 4x3
topology redistribute -name 4x3 -pool AllStorageNodes
topology preview -name 4x3
plan deploy-topology -name 4x3 -wait
```

5. Just as before run the *runadmin.sh* script and pass *4x3.kvs* as the argument.

```
[oracle@bigdatalite scripts]$ ./runScript.sh 4x3.kvs
Executed plan 9, waiting for completion...
Plan 9 ended successfully
Created 4x3
Redistributed: 4x3
Topology transformation from current deployed topology to 4x3:
Create 1 shard
Create 3 RNs
Relocate 2 RNs
Migrate 30 partitions

shard rg2
  Relocate rg2-rn2 from sn2 /tmp/data/sn2/u01 to sn4 /tmp/data/sn4/u02
shard rg3
  Relocate rg3-rn1 from sn1 /tmp/data/sn1/u03 to sn4 /tmp/data/sn4/u01
shard rg4
  3 new RNs : rg4-rn1 rg4-rn2 rg4-rn3
  30 partition migrations

Executed plan 10, waiting for completion...
Plan 10 ended successfully
```

Notice that new storage node has a capacity=3 that is enough for a new shard with RF=3 to be added to the cluster but system determines that if all the RNs from the same shard get hosted on the storage node then it becomes the single point of failure. So it migrates two RNs from two already deployed shards and host the third RN from shard 4. All this migration, rebalancing happens automatically. After the deployment is done you can view the topology from the web console:

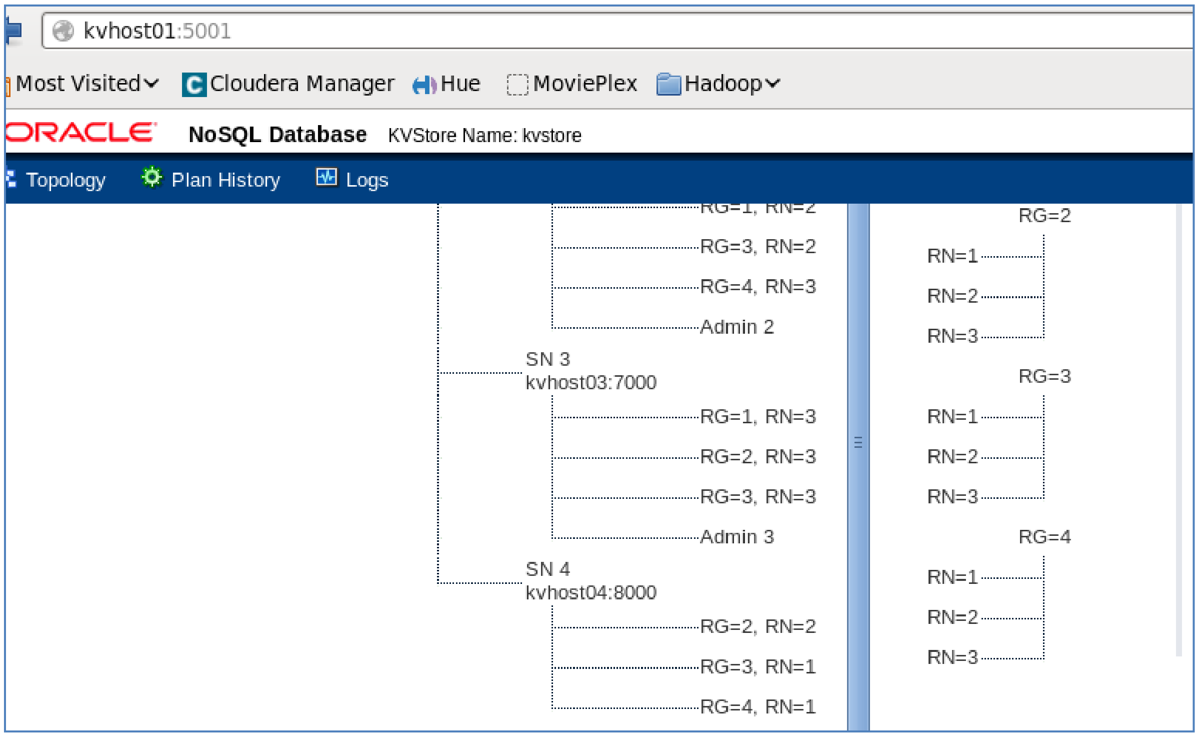


Figure 4: Storage node and all three replication nodes running on that node are from three different shards keeping the cluster HA.

Lab Exercise 3 – Add secondary datacenter to the cluster.

Objective:

Learn how you can add a new datacenter to the existing NoSQL cluster to provide resilience from any disaster scenario.

Summary:

When a new datacenter is added, replication nodes from all the shards get added to this new datacenter, which means each datacenter has the full copy of data. Notice that as a prerequisite you need to have enough capacity to host RNs from each shard. For example we have 4 shards in Houston datacenter at the moment and we can to add new datacenter (Dallas) with RF=1 then at least we need storage node capacity of 4.

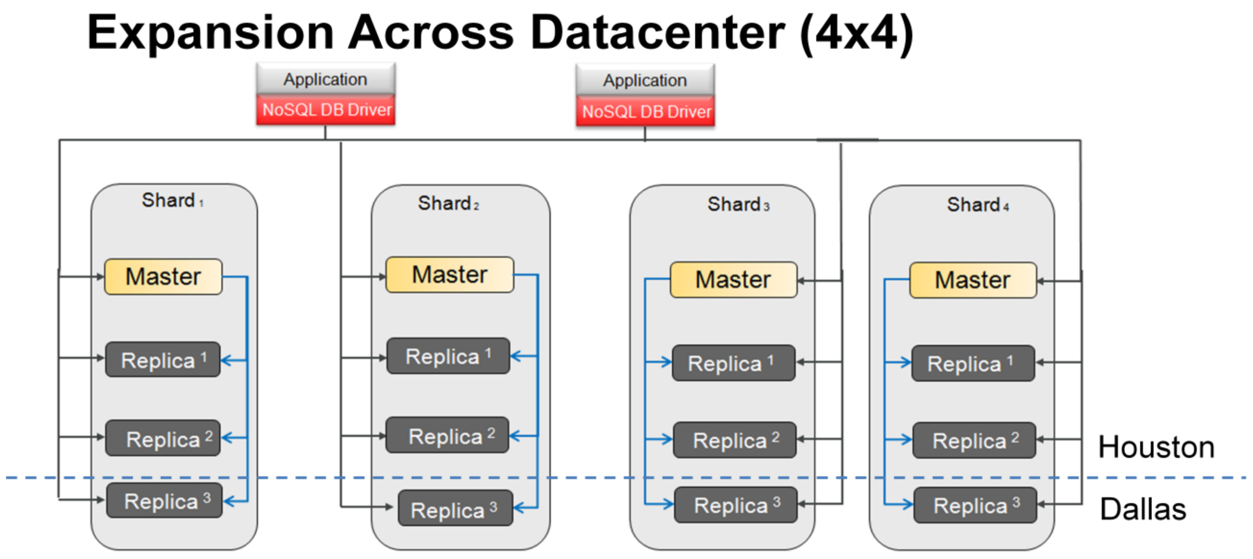


Figure 5: Four shard topology spread across two separate datacenters (Houston & Dallas) with each datacenter having full copy of the data. Houston RF=3 & Dallas RF=1. Combined replication factor of the store = 4

By the end of our third lab we are going to have replication nodes from each of the 4 shards running in Dallas datacenter as well.

Instructions:

1. In production environment you would be adding new physical machines to expand your cluster across different datacenters but we have this VM only in our workshop so we are going to emulate similar condition by creating a new storage node from a different KVROOT and using different ports. Let's view and run makebootconfig-sn5.sh script:

   ```
   [oracle@bigdatalite scripts]$ cat ./makebootconfig-sn4.sh
   #!/bin/bash

   SN5_DATA_HOME=/tmp/data/sn5

   ########### Bootstrap Storage Node 5    #######
   java -jar $KVHOME/lib/kvstore.jar makebootconfig \
           -root $SN5_DATA_HOME/kvroot \
           -capacity 4 \
           -harange 4010,4030 \
           -admin 4001 \
           -port 4000 \
           -memory_mb 200\
           -host kvhost05 \
           -storagedir $SN5_DATA_HOME/u01 \
           -storagedir $SN5_DATA_HOME/u02 \
           -storagedir $SN5_DATA_HOME/u03 \
           -storagedir $SN5_DATA_HOME/u04 \

   echo " Done bootstrapping storage-node 5"
   ##################################################
   ```

   Note: We are going to use port range of 4000 for SN5, as other four SNs are already using 5000, 6000, 7000 & 8000.

2. Bootstrap SN5

   ```
   [oracle@bigdatalite scripts]$ ./makebootconfig-sn5.sh
    Done bootstrapping storage-node 5
   ```

3. Start the agent on newly added storage node:

   ```
   [oracle@bigdatalite scripts]$ cat startSNA-sn5.sh
   java -jar $KVHOME/lib/kvstore.jar start -root /tmp/data/sn5/kvroot &
   ```

```
[oracle@bigdatalite scripts]$ ./startSNA-sn5.sh
```

4. Next we will review 4x4.kvs script. As you can notice that first create a new datacenter Dallas and then add new storage node to Dallas datacenter. Based on the current state of the cluster we will clone the topology and call redistribute command, which will create new shards, replication nodes and move partitions, RN to rebalance the cluster.

```
[oracle@bigdatalite scripts]$ cat 4x4.kvs
### Begin Script ##########################################
plan deploy-datacenter -name "Dallas" -rf 1 -wait

###### Deploy Fifth Storage Node  ########################
plan deploy-sn -dcname "Dallas" -port 4000 -wait -host kvhost05

topology clone -current -name 4x4
topology redistribute -name 4x4 -pool AllStorageNodes
topology preview -name 4x4
plan deploy-topology -name 4x4 -wait
```

5. Just as before run the runadmin.sh script and set 4x4.kvs as the argument.

```
[oracle@bigdatalite scripts]$ ./runScript.sh 4x4.kvs
Executed plan 11, waiting for completion...
Plan 11 ended successfully
Executed plan 12, waiting for completion...
Plan 12 ended successfully
Created 4x4
Redistributed: 4x4
Topology transformation from current deployed topology to 4x4:
Create 4 RNs

shard rg1
  1 new RN : rg1-rn4
shard rg2
  1 new RN : rg2-rn4
shard rg3
  1 new RN : rg3-rn4
shard rg4
  1 new RN : rg4-rn4

Executed plan 13, waiting for completion...
Plan 13 ended successfully
```

Check the latest state of the cluster topology from web console and notice that we now have Dallas datacenter and RN from all 4 shards exist in this DC.

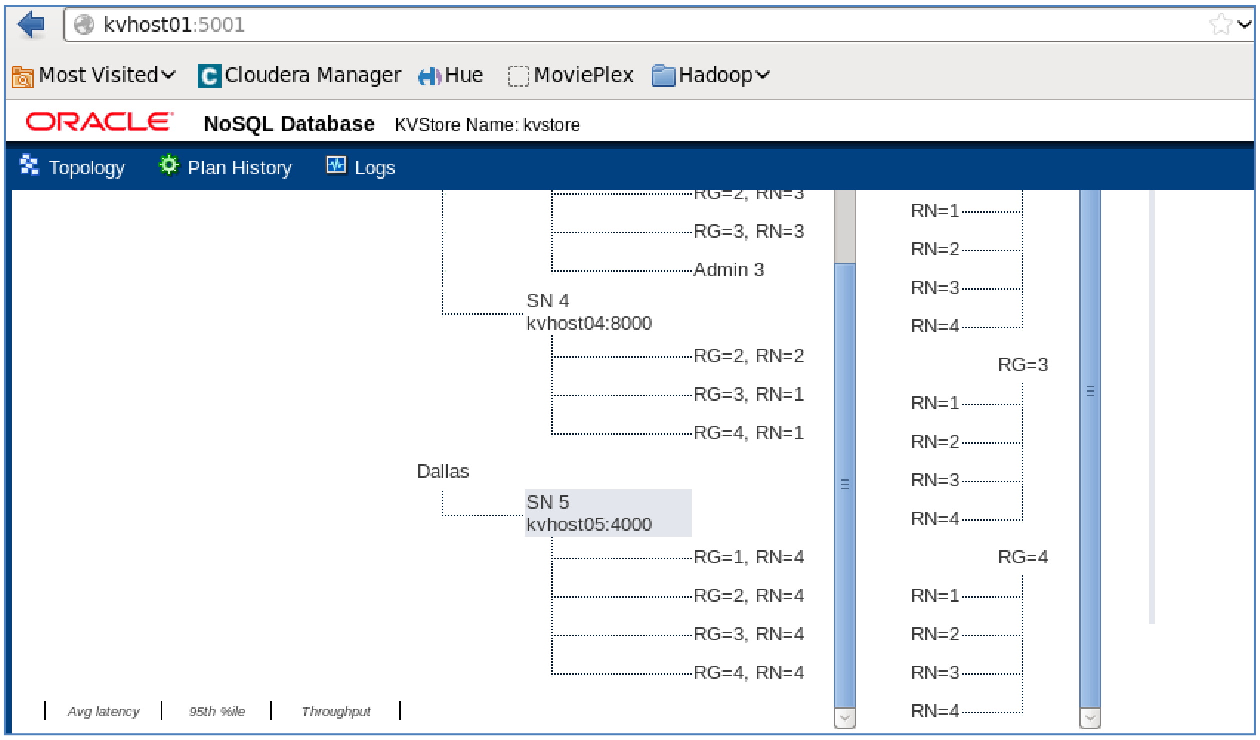


Figure 6: Dallas datacenter added to cluster. Combined replication factor of the cluster is four, of which three are in Houston datacenter and one is in Dallas datacenter.

Lab Exercise 4 – Rolling upgrade of Oracle NoSQL Database cluster.

Objective:

Learn how to upgrade a NoSQL database cluster to a newer version in a rolling upgrade fashion.

Summary:

Upgrading a large distributed cluster of any kind is a daunting effort to say the least. It's never an easy task to make sure that the upgrade process is completely non-intrusive to the business in function. It is also a big challenge to keep the integrity and availability of data while the upgrade is happing.

Oracle NoSQL Database makes it seamless to upgrade a cluster without you figuring out the order or the integrity of the system as whole. Using '*show upgrade*' admin command we will determine what storage nodes can be upgraded in parallel or in serial order without making any shard unavailable for read/write operations.

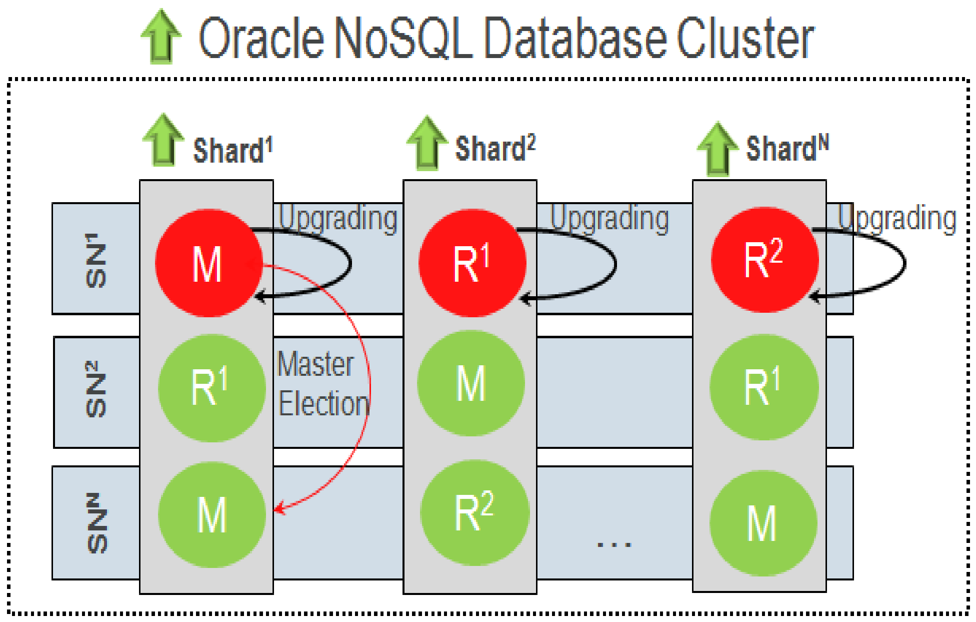


Figure 7: Online rolling upgrade of three storage node cluster. Each storage node has only one replication node (RN) from all the three shards, so bringing one SN down will still maintain the majority and cluster online.

Instructions:

1. To upgrade we need the newer version of Oracle NoSQL Database binaries. To get the latest binaries we need to download it from OTN download page first: http://bit.ly/1mpL2f9

   Note: Download the zip file under /tmp directory

2. Change directory to /u01/nosql and unzip the latest binaries there:

```
[oracle@bigdatalite scripts]$ cd /u01/nosql/
[oracle@bigdatalite nosql]$ ls
kv-2.1.54  scripts
[oracle@bigdatalite nosql]$ unzip /tmp/kv-ee-2.1.57.zip
```

3. Let's now set a new environment variable NEW_KVHOME to the home directory of latest NoSQL binaries.

```
[oracle@bigdatalite nosql]$ export NEW_KVHOME=/u01/nosql/kv-2.1.57/
```

4. With the new variable set let's now change directory to ***scripts*** again. The helper script we are going to run to display the order of upgrade is ***showUpgradeOrder.sh***. Let's review it quick:

```
[oracle@bigdatalite nosql]$ cd scripts
[oracle@bigdatalite scripts]$ cat ./showUpgradeOrder.sh
 ####  Display the new upgrade order
  echo "Latest upgrade Order..."
  java -jar $NEW_KVHOME/lib/kvstore.jar runadmin -port 5000 -host kvhost01 show upgrade
```

   Notice that we are connecting to admin instance using *runadmin* from **NEW_KVHOME** directory**.** This is very important as to find out the order of upgrade we need to run 'show upgrade' command using the latest binaries.

5. Next to display the order of upgrade recommended by the system, we are going to run showUpgradeOrder.sh

```
[oracle@bigdatalite scripts]$ ./showUpgradeOrder.sh

Latest upgrade Order...
Calculating upgrade order, target version: 12.1.2.1.57, prerequisite: 11.2.2.0.23
sn3
sn4
sn1
sn2
sn5
```

Note: We get the storage node id as the output to *show upgrade* command. Storage node in different rows means that those SNs need to be upgrade serially. For example we need to first upgrade sn3 then sn4 then sn1 > sn2 > sn5. SNs that can be upgraded in parallel are displayed in the same row. In our case we don't have SNs that can be upgraded in parallel.

6. Now that we know the order of upgrade let's take a look at how we can upgrade individual storage node. Open script upgradeSNA.sh:

```
[oracle@bigdatalite scripts]$ cat upgradeSNA.sh

…
SN=$1
### Run makebootconfig command ###
if [ -z "$SN" ]
then
  echo "Error: Storage Node ID is not passed as an argument."
  echo "Action: Please rerun the upgrade script by passing storage node id, for example sn1,
sn2 etc"
  exit 1;
else
  ####   Stopping the storage node agent running on given SN ##
  echo "Stopping Storage Node: " $SN
  java -jar $KVHOME/lib/kvstore.jar stop -root /tmp/data/$SN/kvroot &
  #### Sleeping for 5 seconds
  sleep 10

  ###   Starting the storage node agent from new kvhome ######
  echo "Starting Storage Node from NEW_KVHOME: " $NEW_KVHOME
  java -jar $NEW_KVHOME/lib/kvstore.jar start -root /tmp/data/$SN/kvroot &
  #### Sleeping for 10 seconds to give enough time for services to come online ##
  sleep 30
…
```

Note: We first shutdown the storage node by running the stop command form the old KVHOME and then once everything on that storage node is down, we then start the storage node from NEW_KVHOME. That is all to the upgrade process. System will take care of stopping and starting all the managed processes on that storage node. The other thing to also point out is that we need to pass the KVROOT of the Storage Node that we wish to stop/start and because we are using SN-Id (like sn1, sn2 etc) as part of the KVROOT on the same VM, we are passing just the id of the SN as input argument to this script which then construct the full KVROOT path using the SN id and fire the start & stop command.

7. From step 5, remember our order of upgrade is sn3 > sn4 > sn1 > sn2 > sn5, so one by one we will pass these IDs to the *upgradeSNA.sh* script and wait for it to complete.

```
[oracle@bigdatalite scripts]$ ./upgradeSNA.sh sn3

Stopping Storage Node:  sn3
Starting Storage Node from NEW_KVHOME:  /u01/nosql/kv-2.1.57/
Latest upgrade Order...
Redirecting to master at rmi://kvhost03:7000
Calculating upgrade order, target version: 12.1.2.1.57, prerequisite: 11.2.2.0.23
sn4
sn1
sn2
sn5
```

Note: after successfully upgrading the sn3, the script will then display next order of upgrade. And we then use the top most in the list and pass it again to the *upgradeSNA* script. Repeat step 7 until all the SNs are upgraded and at the end you should have your cluster upgraded to latest version of NoSQL Database.

8. If you want to completely automate the whole process to run in a hand free mode then take a look at the script *onlineUpgrade* from **$KVHOME/examples/upgrade** directory

```
[oracle@bigdatalite scripts]$ vi $NEW_KVHOME/examples/upgrade/onlineUpgrade
```

This script will practically run the *show upgrade* command, parse the std-out and then using other admin command will find out where each of the SN is running and will swap the KVHOME to NEW_KVHOME. Exactly what we did in semi automated fashion. This conclude our workshop.

Auxiliary – DBCache tool for sizing.

Objective:

Learn how you can find out what size cluster one need to deploy to store N billion of records of x key size and y record size.

Summary:

For any sizing exercise where the objective is to find cluster size and topology required to support business requirement, it always help to find out how many records do we anticipate in near future and what are the average size of the key and the record. You can then simply plug these numbers to DBCacheSize utitlity that comes bundled in *KVHOME/lib/je.jar*.

```
java -d64 -XX:+UseCompressedOops -jar $KVHOME/lib/je.jar DbCacheSize -records
1000000000 -key 13 -data 1024

=== Database Cache Size ===
 Minimum Bytes    Maximum Bytes   Description
---------------  ---------------  -----------
   32,403,471,488   38,221,653,120  Internal nodes only
1,102,515,920,928  1,108,334,102,560  Internal nodes and leaf nodes
```

Ad in the example we have 1 billion records, with average key size of 13 and average records size of 1KB. The *Internal nodes only* number from the output suggest that to hold 1 billion keys in memory for fast lookup you need 38GB of BDB JE cache. This number is very important because based on how much of RAM is available on each SN, we can find how much more RAM we need or how many of such SN do we need to hold all the keys in BTree stored in memory. Remember when a SN starts we assign 85% of memory available in the system to the java heap which in turn allocate 70% of the memory for the BDB JE.

To quickly find out the net JE cache capacity on any SN just run these number:

> Total RAM available per Storage Node (SN) x 0.85 x 0.70 = 60% of the system memory is allocated for Cache.

i.e. only 60% of the RAM is used by default for JE cache and hence use this number to find out # of SNs required to host required keys. Then other the number to note is '*Internal nodes and leaf node'* that gives us disk storage requirement to store the records of y size (in this case 1KB).